NEURAL INFORMATION PROCESSING SYSTEMS

UNIVERSITY OF TORONTO

PATCH

Learnable Tile-Level Hybrid Sparsity for LLMs

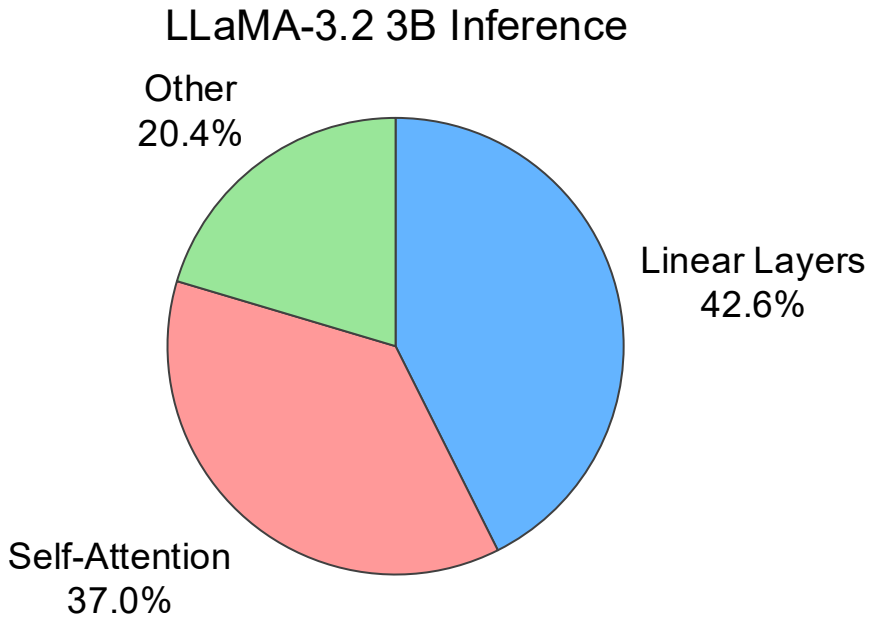
Younes Hourri*, Mohammad Mozaffari*, Maryam Mehri Dehnavi

*Equal Contribution



github.com/Paramathic/patch

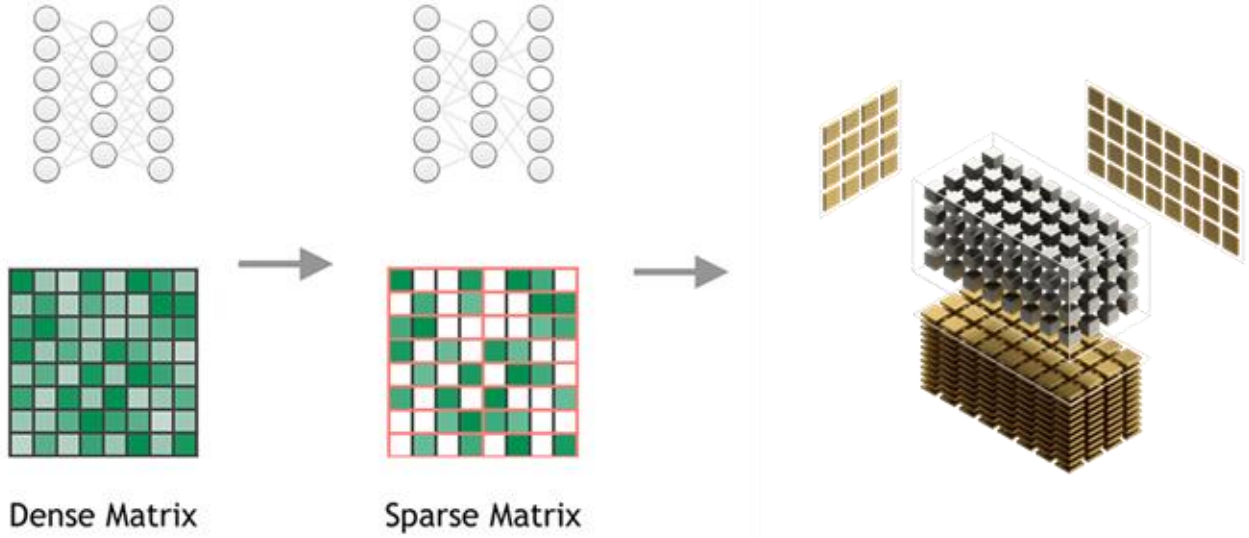
LLM Compute Graph | Time Breakdown



Linear Layers are the bottleneck at inference!

Weight Sparsity | Sparse Tensor Cores

We focus on 2:4 sparsity because of the available hardware support!



2:4 Sparsity → 2 out of 4 consecutive elements are zero

Sparsity Challenges

Unstructured Sparsity: Preserves accuracy but is hard to accelerate

2:4 Sparsity: Fixed 50% in every layer, with a large accuracy gap

LLaMA-2 7B Average Accuracy on 8 Zero-shot Tasks*

Method	Average Accuracy
Dense	54.61%
50% 2:4, best one-shot**	45.92% (84%)
50% 2:4, learned (MaskLLM)	48.62% (89%)

*MMLU, PIQA, ARC-Easy, ARC-Challenge, WinoGrande, OpenBookQA, RACE, HellaSwag; parentheses: share of dense accuracy

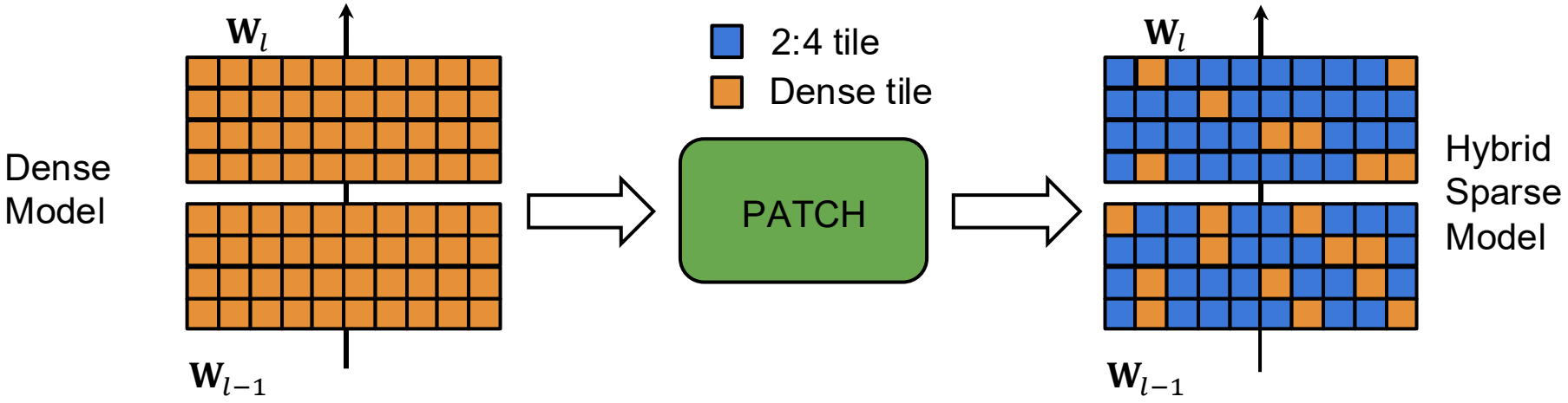
**ProxSparse, the best of Magnitude, Wanda, SparseGPT, Thanos, and ProxSparse

PATCH

Hybrid approach

Combines the flexibility of unstructured with the efficiency of 2:4

Jointly learns a mask that selects dense vs. sparse tiles, and the 2:4 pattern within sparse tiles



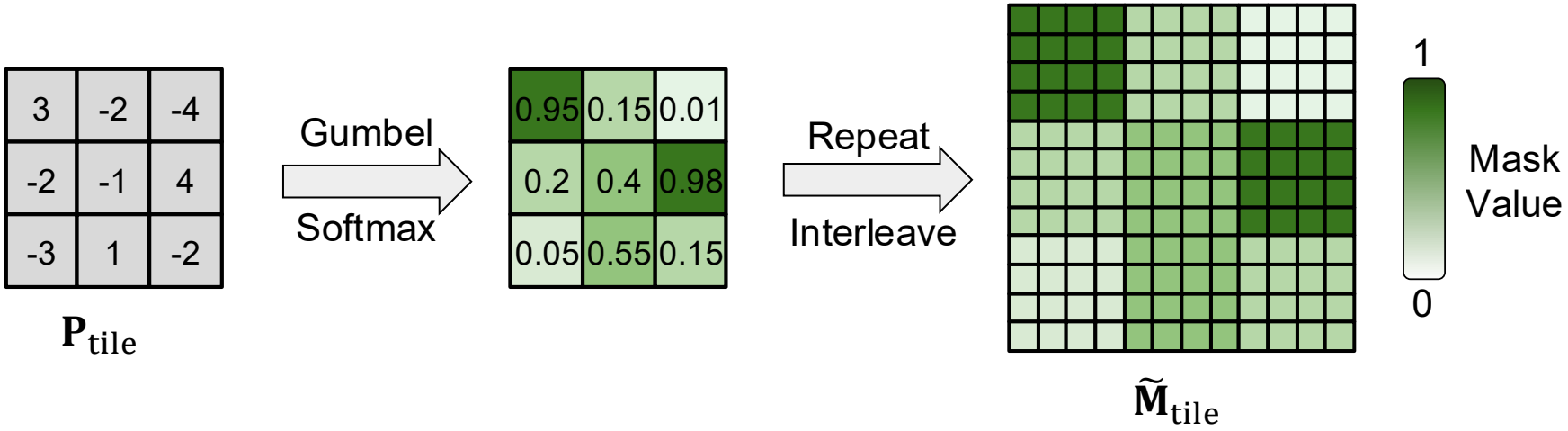
PATCH | Tile Selection

$\mathbf{P}_{\text{tile}}$: Tile logits
$\widetilde{\mathbf{M}}_{\text{tile}}$: Sampled Tile Mask

We frame tile selection as a probabilistic sampling problem :

Using Gumbel-Softmax (GS), we turn logits into probabilities and sample sparsity patterns

The randomness will allow us to explore the space of mask combinations



PATCH | Sparse Tile 2:4 Pattern Selection

2 options

- Use a fixed 2:4 pattern generated beforehand (e.g. one-shot methods)

  Memory efficient (PATCH$^{\text{Tile}}$)
  Can train 8B model on a single 80GB GPU



Frozen

- Learn the 2:4 pattern during training

  Better flexibility and performance (PATCH$^{\text{Joint}}$)



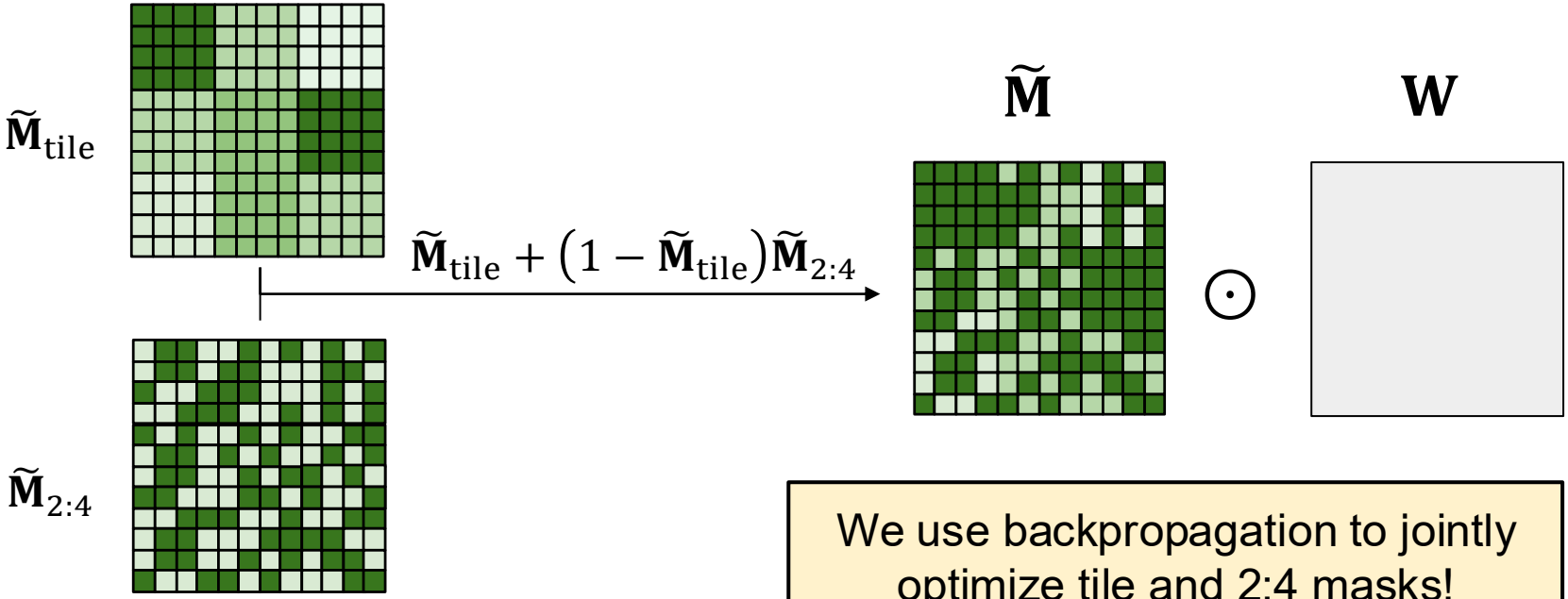
Learnable

PATCH | Combining Masks

$\widetilde{\mathbf{M}}_{2:4}$: Sampled 2:4 mask
$\widetilde{\mathbf{M}}_{\text{tile}}$: Sampled Tile Mask
$\mathbf{W}$: Weight

The tile mask is combined with the 2:4 mask to form the final sparsity pattern

Each tile is a weighted average of dense and 2:4, controlled by $\widetilde{\mathbf{M}}_{\text{tile}}$



We use backpropagation to jointly optimize tile and 2:4 masks!

PATCH | Training Objective

ρ : Target Density
$\widetilde{\mathbf{M}}$: Sampled Mask
$\mathbf{W}$: Weight

$$\mathcal{L} = \mathcal{L}_{\text{LM}}\left(x; \{\widetilde{\mathbf{M}}_i \odot \mathbf{W}_i\}_i\right) + \lambda_1 \left| \frac{\sum_i \|\widetilde{\mathbf{M}}_i\|_1}{\sum_i \|\mathbf{W}_i\|_0} - \rho \right| - \lambda_2 \frac{\sum_i \|\widetilde{\mathbf{M}}_i \odot \mathbf{W}_i\|_2^2}{\sum_i \|\mathbf{W}_i\|_2^2}$$

Language Modelling Loss

(Increase Model Quality)

Sparsity Regularizer

(Fix Model Sparsity Ratio)

Weight Regularizer

(Improve Training Stability)

PATCH^{Joint} Results

At 45% sparsity, PATCH beats 50% 2:4 MaskLLM on every model.

Average Accuracy on 8 Zero-shot Tasks*

Sparsity	Method	Qwen-2.5 0.5B	LLaMA-3.2 1B	Gemma-3 1B	Average Gain over MaskLLM (points)
0%	Dense	46.00	47.70	47.01	
50%	MaskLLM (2:4)	39.33 (86%)	41.04 (86%)	41.84 (89%)	
45%	PATCH	40.29 (88%)	42.08 (88%)	42.80 (91%)	+0.99
35%	PATCH	41.15 (89%)	42.72 (90%)	43.30 (92%)	+1.65
25%	PATCH	42.39 (92%)	43.81 (92%)	44.07 (94%)	+2.69

*MMLU, PIQA, ARC-Easy, ARC-Challenge, WinoGrande, OpenBookQA, RACE, HellaSwag; parentheses: share of dense accuracy

PATCH^{Tile} Results

Learning only the tiles carries the trade-off to larger models.

Average Accuracy on 8 Zero-shot Tasks*

Sparsity	Method	LLaMA-2 7B	LLaMA-3.1 8B
0%	Dense	54.61	60.31
50%	MaskLLM (2:4)	48.62 (89%)	52.80 (88%)
45%	PATCH	48.99 (90%)	53.60 (89%)
35%	PATCH	50.08 (92%)	55.28 (92%)
25%	PATCH	51.58 (94%)	56.48 (94%)

LLaMA-2 13B, 45% sparsity
91% of dense accuracy vs.
87% for ProxSparse (50% 2:4)

Average Gain over MaskLLM (points)

→ **+0.59**

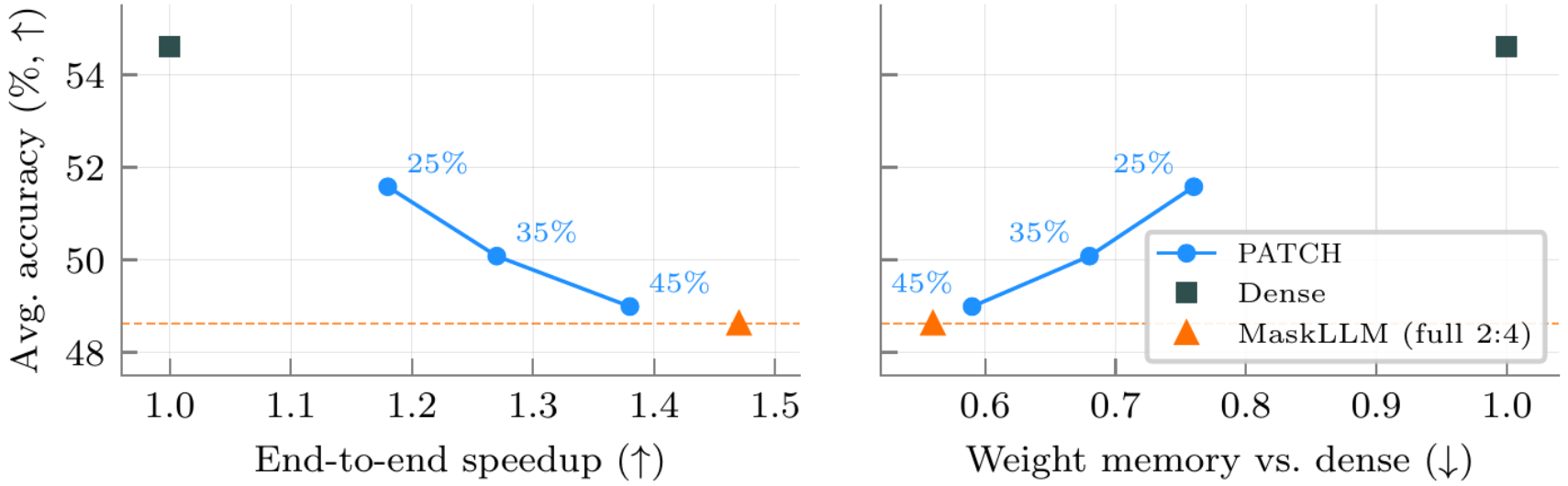
→ **+1.97**

→ **+3.32**

*MMLU, PIQA, ARC-Easy, ARC-Challenge, WinoGrande, OpenBookQA, RACE, HellaSwag; parentheses: share of dense accuracy

Speedup and Memory Savings

LLaMA-2 7B on an A6000, batch size 16, executed with the STOICC compiler



1.38x faster and **0.59x weight memory** at 45% sparsity; every PATCH point is more accurate than full 2:4

Prefill and generation lengths of 128 tokens; prefill and decoding captured with CUDA graphs

Conclusion

PATCH enables continuous sparsity from 0% to 50%

Dense and 2:4 tiles bridge unstructured and 2:4 pruning

88% to 96% of dense accuracy at 25% to 45% sparsity, up to 13B parameters

Up to 1.38x end-to-end speedup on an A6000 GPU



Code : github.com/Paramathic/patch
Checkpoints : huggingface.co/collections/mohammad-mozaffari/patch



Paper : arxiv.org/abs/2509.23410